DATA C88C
Spring 2025

Michael Ball
Final

Solutions last updated: Monday, November 10, 2025

Your name: _____

Your student ID: _____

Your Berkeley email: _____

Your room location: _____

Student ID of the person to your left: _____

Student ID of the person to your right: _____

You have 180 minutes. There are 9 questions of varying credit. (75 points total)

Question:	HC	1	2	3	4	5	6	7	8	9	Total
Points:	1	10	11	9	7	8	11	5	13	0	75

For questions with **circular bubbles**, you may select only one choice.

- ○ Unselected option (Completely unfilled)
- ◌ Don't do this (it will be graded as incorrect)
- ● Only one selected option (completely filled)

For questions with **square checkboxes**, you may select one or more choices.

- ■ You can select
- ■ multiple squares
- ☑ (Don't do this)

Anything you write outside the answer boxes or you ~~cross out~~ will not be graded. If you write multiple answers, your answer is ambiguous, or the bubble/checkbox is not entirely filled in, we will grade the worst interpretation. For coding questions with blanks, you may write at most one statement per blank and you may not use more blanks than provided.

As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others. I will follow the rules of this exam.

I have read and agree to the honor code above.

(1 point) Sign your name: _____

Q1 *Multifandom Potpourri* **(10 points)**

Q1.1 (1 point) What would Python display?

```
>>> champs = ["Vi", "Jinx", "Caitlyn", "Ekko", "Jayce", "Viktor"]
>>> hextech = [champs[c] for c in range(len(champs)) if c % 2 == 0]
>>> sum(map(len, hextech))
```

○ 6 ● 14 ○ 28 ○ 39

Q1.2 (2 points) Select all of the following statements that are true about object-oriented programming.

☐ Objects and classes are 2 words for the same thing

■ When implementing the `__init__` method, there is no explicit `return` statement

■ Class attributes can be accessed through the class name or through `self`

■ When implementing a `Square` and `Rectangle` class, a `Square` could inherit from `Rectangle`

☐ When implementing an `Account` and `Bank` class, an `Account` could inherit from `Bank` because an account is a type of bank.

Solution: Objects are *instances* of classes — they are not the same thing.

The `__init__` method does not have an explicit return, but when it is called it will return the newly created instance.

Inheritance describes an *is-a* relationship while *composition* describes a *has-a* relationship.

Because of OOP lookup rules, if there is a class attribute called `foo` and Python is trying to evaluate `self.foo`, it will first look for an instance attribute/method called `foo`, and if it doesn't find one, it will look for a class attribute/method called `foo`. Using the class name forces Python to look directly for a class attribute called `foo`.

Since all squares are rectangles, `Square` should inherit from `Rectangle`, not the other way around.

For Q1.3 - Q1.4

Suppose we have defined the following code:

```
def white_lotus(group):
  def hotel(thailand):
      lotus_iter = iter(group)
      i = 0
      while i < thailand:
        print(next(lotus_iter))
        i += 1
  return hotel

group = ["Kate", "Laurie", "Jaclyn"]
```

Q1.3 (2 points) What would Python display? Write each printed line on its own line. If an error occurs after printing, write `Error` on a new line.

```
>>> white_lotus(group)(2)
```

Kate
Laurie

Q1.4 (2 points) What would Python display? Write each printed line on its own line. If an error occurs after printing, write `Error` on a new line.

```
>>> white_lotus(group)(5)
```

Kate
Laurie
Jaclyn
Error

Solution: Since `thailand` is 5, the `while` loop will call `next` more times than the length of `group`, causing a `StopIteration` error to be raised after the names are printed.

Q1.5 (1 point) What would Python display?

```
1 >>> ellie = { 4: 5, 2: 3, 6: 1 }
2 >>> joel = lambda x: x + ellie[x]
3 >>> max(ellie, key=joel)
```

- ○ 1
- ● 4
- ○ 2
- ○ 5
- ○ 3
- ○ 6

Solution: The `key` parameter of the `max` function is called on each element of the iterable passed in (`ellie`) and its return value is used to sort the elements. Since the key function `joel` returns sum of the current key-value pair in the dictionary and 4 + 5 is the largest sum, the corresponding key (4) is returned.

Q1.6 (1 point) What would Python display?

```
1 >>> ('mothma' and 0) or 'luthen' or 5 / 0
```

- ○ `True`
- ○ `0`
- ○ `False`
- ● `'luthen'`
- ○ `'mothma'`
- ○ A `ZeroDivisionError` would occur

Solution: First `'mothma' and 0` is evaluated and returns `0` since `0` is falsy. Then `0 or 'luthen'` is evaluated and returns `'luthen'` since a non-empty string is truthy. Then `'luthen' or 5 / 0` is evaluated, but it short circuits after `'luthen'` and this is what is returned. This means `5 / 0` is never evaluated and an error doesn't occur.

Q1.7 (1 point) Recall the `every_other` function from Discussion 8. It should mutate a linked list so that all the elements with odd indices are removed (using 0-based indexing).

Below is a **buggy implementation** of `every_other`.

```
1  def every_other(s):
2    """
3    >>> s = Link(1, Link(2, Link(3, Link(4))))
4    >>> every_other(s)
5    >>> s
6    Link(1, Link(3))
7    >>> odd_length = Link(5, Link(3, Link(1)))
8    >>> every_other(odd_length)
9    >>> odd_length
10   Link(5, Link(1))
11   >>> singleton = Link(4)
12   >>> every_other(singleton)
13   >>> singleton
14   Link(4)
15   """
16   if s.rest is Link.empty or s is Link.empty:
17       return
18   else:
19       s.rest = s.rest.rest
20       every_other(s.rest)
```

When the doctests are run, we get the following error:

```
AttributeError: 'tuple' object has no attribute 'rest'
```

What should be changed in order to fix the implementation?

Hint: It may be useful to reference the `Link` class on the reference sheet.

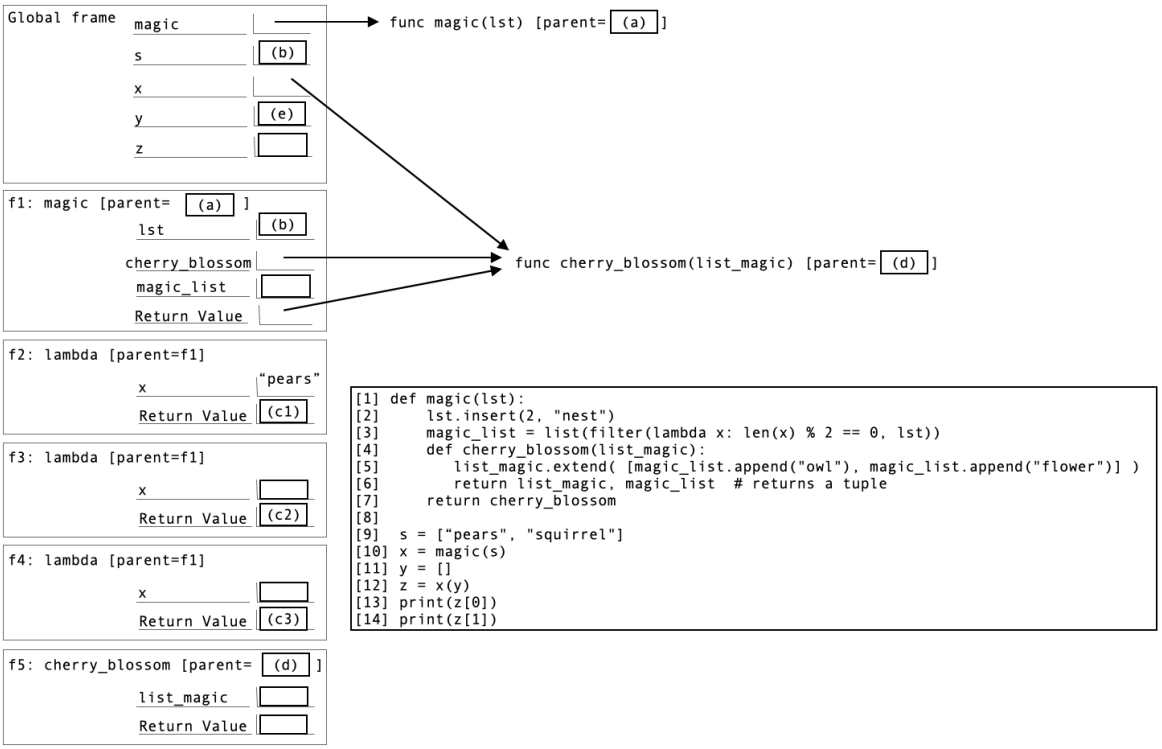
- ● Change line 16 to `if s is Link.empty or s.rest is Link.empty:`
- ○ Change line 17 to `return Link.empty`
- ○ Change line 19 to `s = s.rest`
- ○ Change line 20 to `return every_other(s.rest)`

Solution: The error indicates that we are accidentally trying to access the `rest` attribute from an empty linked list (since `Link.empty` is defined as an empty tuple in the `Link` class). This is because we first check if `s.rest is Link.empty`, but it's possible that `s` is already empty. Thus, the fix is to swap the order in which we check the conditions.

Q2 *Mutating Magic* **(11 points)**

Fill in the blanks to complete the environment diagram. Assume code has been fully run before filling in blanks. The blanks with no labels have no questions associated with them and are not scored. They are hidden from the environment diagram for the purpose of assessment, but there should be content in those blanks. Please note: If more than one blank shares the same label (e.g., `(d)`), they have the same answer.

Note: The `insert` list method does not error if the input index is equal to the length of the list.



Q2.1 (1 point) Fill in blank (a).

- ● `Global`
- ○ `f1`
- ○ `f2`
- ○ `f3`

Q2.2 (1 point) Fill in blank (b).

- ○ `['squirrel', 'pears', 'nest']`
- ○ `['pears', 'squirrel']`
- ● `['pears', 'squirrel', 'nest']`
- ○ `['pears', 'nest']`
- ○ `['squirrel', 'nest', 'pears']`

Q2.3 (2 points) What are the values of `c1`, `c2`, and `c3`, respectively? Please provide your answers in that order.

- ○ `False, False, False`
- ○ `False, True, False`
- ○ `True, True, True`
- ● `False, True, True`
- ○ `True, True, False`

Q2.4 (1 point) Fill in blank (d).

- ○ `Global`
- ● `f1`
- ○ `f2`
- ○ `f3`

Q2.5 (2 points) Fill in blank (e).

[None, None]

Q2.6 (2 points) What does line 13 print out?

[None, None]

Q2.7 (2 points) What does line 14 print out?

['squirrel', 'nest', 'owl', 'flower']

Q3 *Interstellar Fleet* **(9 points)**

Captain Nova is organizing her fleet for an interstellar mission. Help her implement the class `Rocket` that inherits from the `Spacecraft` class. The base class is defined as follows:

```
class Spacecraft:
    """
    >>> s = Spacecraft("Fondor", 1000)
    >>> s.craft_type
    'Fondor'
    >>> s.payload_capacity
    1000
    >>> s.status_report()
    All systems nominal
    """
    def __init__(self, craft_type, payload_capacity):
        self.craft_type = craft_type
        self.payload_capacity = payload_capacity

    def status_report(self):
        print("All systems nominal")
```

Q3.1 - Q3.3 (4 points) Complete the constructor for the `Rocket` class so that it initializes the same attributes as `Spacecraft`, in addition to two new attributes: `name` and `thrust`.

```
class Rocket(Spacecraft):
    """
    >>> r1 = Rocket("Apollo", "Orbiter", 1000, 3000)
    >>> r1.name
    'Apollo'
    >>> r1.craft_type
    'Orbiter'
    >>> r1.payload_capacity
    1000
    >>> r1.thrust
    3000
    """
    def __init__(self, name, craft_type, payload_capacity, thrust):
        super().__init__(craft_type, payload_capacity)
                                Q3.1
        self.name = name
               Q3.2
        self.thrust = thrust
                  Q3.3

    def status_report(self):
        print("Rocket systems nominal")
```

Q3.4 (1 point) What would the lines below print?

```
>>> r1 = Rocket("Apollo", "Orbiter", 1000, 3000)
>>> r1.status_report()
```

```
Rocket systems nominal
```

Q3.5 (2 points) Implement the method `calculate_fuel_needed` for the `Rocket` class. Assume that fuel needs are calculated based on the following rules:

1. 0.5 units of fuel per unit of `payload_capacity`, **and**
2. 0.1 units of fuel per unit of `thrust`

```
def calculate_fuel_needed(self):
    """
    >>> r1 = Rocket("Apollo", "Orbiter", 1000, 3000)
    >>> r1.calculate_fuel_needed()
    800.0
    """

    return self.payload_capacity * 0.5 + self.thrust * 0.1
```
Q3.5

Q3.6 (1 point) Captain Nova wants to compare two different rocket configurations. Write a method `is_more_powerful_than` for the `Rocket` class that takes another rocket, `other_rocket`, as an input and returns `True` if the current rocket's `thrust` is strictly greater than the other rocket's `thrust`, and `False` otherwise.

```
def is_more_powerful_than(self, other_rocket):
    """
    >>> r1 = Rocket("Apollo", "Orbiter", 1000, 3000)
    >>> r2 = Rocket("Artemis", "Orion", 100, 500)
    >>> r1.is_more_powerful_than(r2)
    True
    """

    return self.thrust > other_rocket.thrust
```
Q3.6

Q3.7 (1 point) Suppose you want every rocket to automatically report its fuel needs when it launches. Write the `launch_sequence` method in the `Rocket` class so that prints `"Fuel required: <fuel>"`. Assume that `calculate_fuel_needed` has been implemented correctly. Your solution must minimize the amount of repeated code.

```
def launch_sequence(self):
    """
    >>> r1 = Rocket("Apollo", "Orbiter", 1000, 3000)
    >>> r2 = Rocket("Artemis", "Orion", 100, 500)
    >>> r1.launch_sequence()
    Fuel required: 800.0
    >>> r2.launch_sequence()
    Fuel required: 100.0
    """

    print(f"Fuel required: {self.calculate_fuel_needed()}")
```

Q3.7

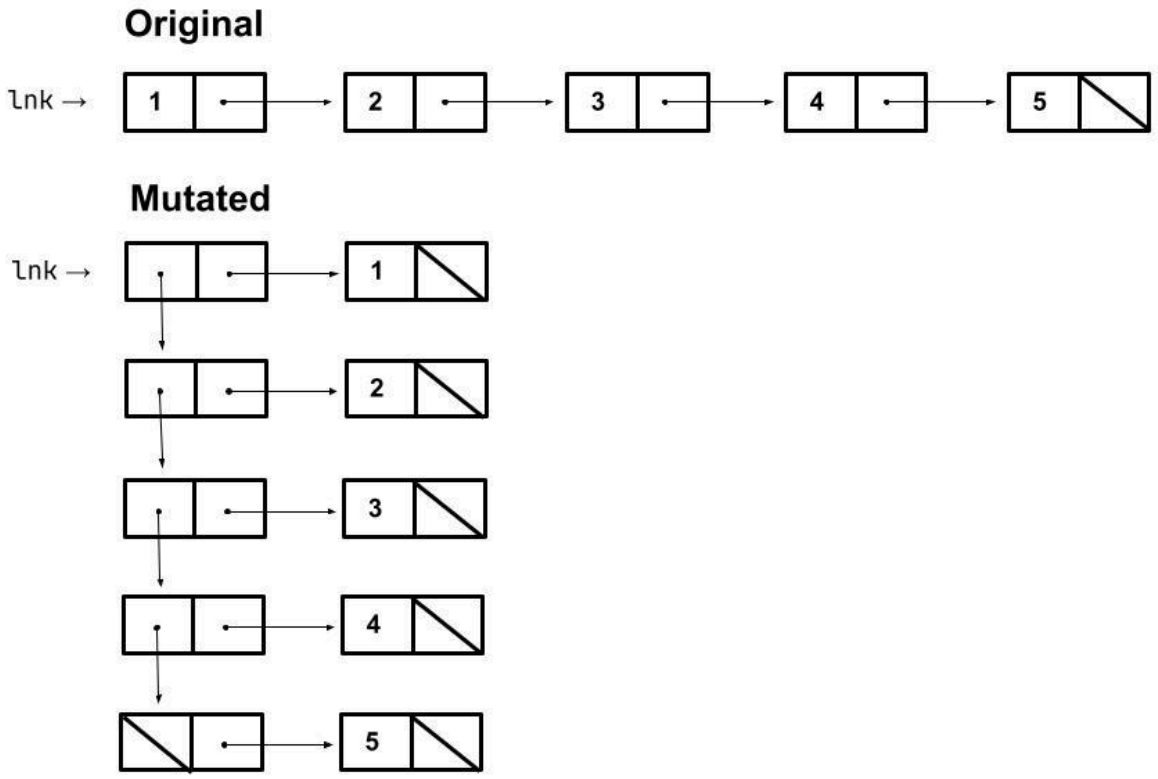
Solution: There are many acceptable answers, such as `print("Fuel required:", self.calculate_fuel_needed())` or using string concatenation: `print("Fuel required: " + str(self.calculate_fuel_needed()))`.

Q4 *Linked List Vectors* **(7 points)**

Your job is to implement the function `transpose_lnk`, which takes in a linked list `lnk` and **mutates** it such that it becomes transposed. A transposed object, such as a matrix or vector, is one where (in the context of this problem) the horizontal structure of the object becomes vertical (see the diagram).

When the linked list is transposed, each node's rest attribute now points to a new linked list node containing the original value. The original link's `first` attribute is replaced with the `rest` of the transposed linked list. You may assume all values inside the passed in linked list are integers.

The top linked list represents the original linked list, and the bottom linked list represents the transposed linked list.



Q4.1 - Q4.5 (7 points) Implement the function below.

Note: Assume that `check_lnk` is implemented already and returns `True` if the first linked list is equal to the second one.

```
def transpose_lnk(lnk):
  """
  >>> lnk = Link(1, Link(2, Link(3, Link(4, Link(5)))))
  >>> five = Link(Link.empty, Link(5))
  >>> four = Link(five, Link(4))
  >>> three = Link(four, Link(3))
  >>> two = Link(three, Link(2))
  >>> expected = Link(two, Link(1))
  >>> # The function should not make a new linked list!
  >>> print(transpose_lnk(lnk))
  None
  >>> check_lnk(lnk, expected)
  True
  """
  current = lnk
  while current is not Link.empty:
    val = current.first
          Q4.1
    current.first = current.rest
                    Q4.2
    current.rest = Link( val )
    Q4.3               Q4.4
    current = current.first
              Q4.5
```

Q5 *Pine-ing for Feedback* **(8 points)**

C88C Staff was reviewing responses from the Mid-Semester Feedback Form when a rogue Stanford student tampered with the responses — adding extra words and rearranging sentences into trees! Staff now needs your help recovering valid feedback.

Implement the function `contains_phrase` which takes in two parameters:

1. A tree `feedback` where each node is a string containing a single word
2. A list of words `target` that represents the original response staff wants to find. Any other word not contained in `target` was added by the Stanford student.

The function returns `True` if every word in `target` appears in `feedback` in order continuously (i.e. along the same branch), without the extra words added by the Stanford student. Refer to the doctests and diagrams below for handling examples. Otherwise, return `False`.

Notes:

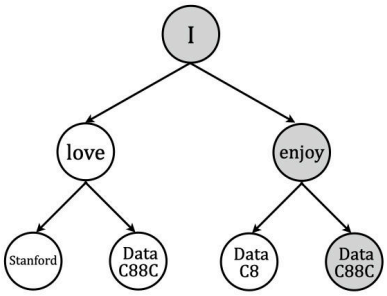
- In the diagram and doctests below, we consider `'DataC88C'` and `'DataC8'` to each be single words for simplicity
- The phrase can start at any node (not only the root node)
- The phrase can end at any node (not only a leaf node)
- Assume `feedback` is non-empty
- Assume `len(target) > 1` (e.g. you do not need to handle the cases where `target` is empty or contains exactly 1 word)

`feedback` trees that return `True` with `target ["I", "enjoy", "DataC88C"]`

Contains the target phrase continuously in order (i.e. along the same branch):

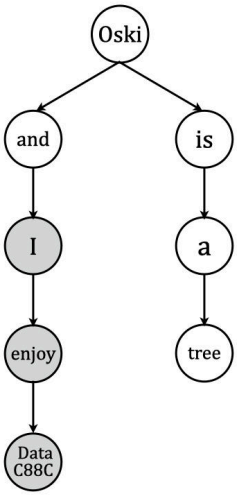
t1:

t2:



Target phrase does NOT have to start at the root or end at a leaf

t3:

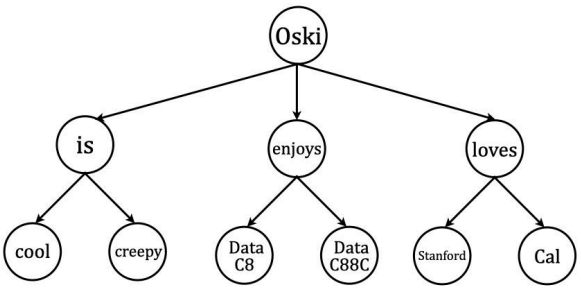


t4:

`feedback` trees that return **`False`** with **`target ["I", "enjoy", "DataC88C"]`**

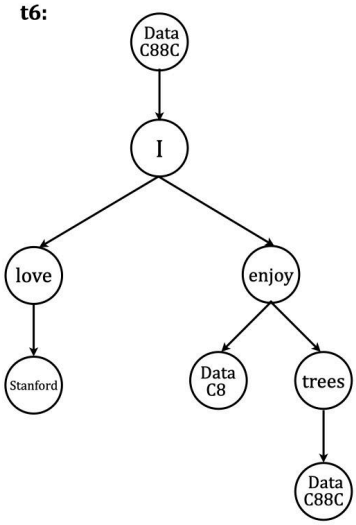
Does not contain the target phrase

t5:



All strings in the target phrase must appear uninterrupted in the tree

t6:




```
1  def contains_phrase(feedback, target):
2      """
3      Most doctests omitted due to length, see the diagram above!
4      >>> target = ['I', 'enjoy', 'DataC88C']
5      >>> t1 = Tree('I', [Tree('enjoy', [Tree('DataC88C')])])
6      >>> contains_phrase(t1, target)
7      True
8      """
9      if feedback.is_leaf():
10         return len(target) == 1 and ____(a)____
11     elif ____(b)____:
12         return feedback.label == target[0]
13
14     for b in feedback.branches:
15         if ____(c)____ and contains_phrase(____(d)____):
16             return True
17         elif contains_phrase(____(e)____):
18             return True
19     return False
```

Q5.1 (2 points) Fill in blank (a).

feedback.label == target[0]

Solution: In the case where the current **`Tree`** node is a leaf, we must ensure that we are down to the last word in the **`target`** list and that the last word in the list matches.

Q5.2 (2 points) Fill in blank (b).

len(target) == 1

Q5.3 (2 points) Fill in blank (c).

feedback.label == target[0]

Solution: If there is more than 1 word left in `target`, we must recursively search through the branches of `feedback`. If the current branch's word matches the current target word, we should recurse on that branch and look for the rest of the target words, not including the current word.

Q5.4 (1 point) Fill in blank (d), the arguments to `contains_phrase`.

- ◯ `feedback, target[0]`
- ◯ `b, target`
- ◯ `b, target[0]`
- ● `b, target[1:]`
- ◯ None of the above

Solution: See explanation for part (c).

Q5.5 (1 point) Fill in blank (e), the arguments to `contains_phrase`.

- ◯ `feedback, target[0]`
- ● `b, target`
- ◯ `b, target[0]`
- ◯ `b, target[1:]`
- ◯ None of the above

Solution: If there is more than 1 word left in `target`, we must recursively search through the branches of `feedback`. If the current branch's word does NOT match the current target word, we should recurse on that branch and continue looking for the current target word, since the phrase can begin anywhere in the tree.

Q6 *Animal Party* **(11 points)**

A group of animals is getting together to socialize! The `AnimalMeetings` class is an iterator that takes in a list of `(name, breed)` tuples via its `__init__` method. Complete the `__next__` method so that it iterates through the list and returns a string describing each animal's introduction at the party. When `next` is called on the iterator, it should return all **unique** pairs of animals of **different** breeds.

Notes:

- Each pair should be returned only once regardless of order, meaning you should **NOT** return both "Linda the Lion meets Jenny the Jellyfish" and "Jenny the Jellyfish meets Linda the Lion"
- Your implementation must iterate through the list from beginning to end, producing pairs in that order.

Hint: As you build unique pairs, think about how to systematically move through the list without repeating or flipping combinations. How might the relationship between your two indices help you do that? What conditions help you avoid going out of bounds?

```
class AnimalMeetings:
  """
  >>> animals = [
  ...     ("Linda", "Lion"),
  ...     ("Millie", "Monkey"),
  ...     ("Jenny", "Jellyfish"),
  ...     ("Laila", "Lion")
  ... ]
  >>> animal_iter = AnimalMeetings(animals)
  >>> next(animal_iter)
  "Linda the Lion meets Millie the Monkey"
  >>> next(animal_iter)
  "Linda the Lion meets Jenny the Jellyfish"
  >>> next(animal_iter)
  "Millie the Monkey meets Jenny the Jellyfish"
  >>> next(animal_iter)
  "Millie the Monkey meets Laila the Lion"
  """

  def __init__(self, animals):
    self.animals = animals
    self.first = 0
    self.second = 1

  def __iter__(self):
    return self
```

```
def __next__(self):
  while ______(a)______:
    while self.second < len(self.animals):
      name1, breed1 = self.animals[self.first]
      name2, breed2 = self.animals[self.second]
      self.second = ______(b)______
      if ______(c)______:
        ______(d)______
    self.first += 1
    self.second = ______(e)______
  raise ______(f)______
```

Q6.1 (2 points) Fill in blank (a).

self.first < len(self.animals) - 1

Q6.2 (2 points) Fill in blank (b).

self.second + 1

Q6.3 (2 points) Fill in blank (c).

breed1 != breed2

Q6.4 (1 point) Select which response belongs in blank (d).

Hint: Imagine `x = "Good"` and `y = "Luck"`. The syntax `f"{x} {y}!"` would give us the sentence `"Good Luck!"`

- ● `return f"{name1} the {breed1} meets {name2} the {breed2}"`
- ○ `print(f"{name1} the {breed1} meets {name2} the {breed2}")`
- ○ `self.next = f"{name1} the {breed1} meets {name2} the {breed2}"`
- ○ None of the above

Q6.5 (1 point) Fill in blank (e).

self.first + 1

Solution: We must reset `self.second` to the item after `self.first` so that each animal is only paired with animals that come after it in the list. This avoids duplicates and self-pairings, and ensures we generate each unique pair exactly once.

Q6.6 (1 point) Fill in blank (f).

StopIteration

Q6.7 (2 points) One animal likes lions more than other breeds of animals. They modify the `__next__` method so that when a lion is encountered, that lion's name and breed is appended to the `animals` list. See the modified function below.

```
def __next__(self):
  while ______(a)______:
    while self.second < len(self.animals):
        name1, breed1 = self.animals[self.first]
        name2, breed2 = self.animals[self.second]
        self.second = ______(b)______
        if breed1 == "Lion": # only modification
          animals.append((name1, breed1))
        if ______(c)______:
          ______(d)______
    self.first += 1
    self.second = ______(e)______
  raise ______(f)______
```

What would the result be if we created an `AnimalMeetings` instance using a list with at least one lion in it?

- ○ The function would run as expected, equivalent to the original `AnimalMeetings` class.
- ○ Per extra Lion, an extra pair of animals would be returned.
- ● An infinite iterator would be created.
- ○ A `StopIteration` error would be raised immediately.

Solution: Because a new lion is added each time a lion is encountered, when we have an input list that contains a lion, the input list of `animals` will grow infinitely. Because the outer and inner loops are both based on `len(animals)` and `len(animals)` increases with each addition of a lion, the loop will never terminate and each call of `__next__` will return an additional pair of animals.

Q7 *Fun Times with Runtimes* **(5 points)**

For all of the subparts in this question, select the runtime of the function `f` with respect to the size of the input `n`. You may assume `n` is a large positive integer.

Q7.1 (1 point)

```
def f(n):
  result = 0
  for i in range(n // 3):
    result += i
  return result
```

- ○ $O(1)$
- ● $O(n)$
- ○ $O(\log(n))$
- ○ $O(n\log(n))$
- ○ $O(n^2)$
- ○ $O(2^n)$

Solution: The loop will run $\frac{n}{3}$ times, but since we are dealing with big O notation, multiplicative constants are dropped, so `f` still runs in $O(n)$ time.

Q7.2 (1 point)

```
def f(n):
  return sum([i for i in range(5)])
```

- ● $O(1)$
- ○ $O(n)$
- ○ $O(\log(n))$
- ○ $O(n\log(n))$
- ○ $O(n^2)$
- ○ $O(2^n)$

Solution: The loop inside the list comprehension always runs 5 times regardless of the size of `n`, so the program runs in constant time.

Q7.3 (1 point)

```
def f(n):
  i = 0
  while i < n:
    j = 0
    while j < n:
      j += 1
    i += 1
```

- ○ $O(1)$
- ○ $O(n)$
- ○ $O(\log(n))$
- ○ $O(n\log(n))$
- ● $O(n^2)$
- ○ $O(2^n)$

Solution: The outer loop runs `n` times and each inner loop runs `n` times. Thus, the total runtime is $O(n * n) = O(n^2)$

Q7.4 (1 point)

```
def f(n):
  i = 0
  j = 0
  while i < n:
    while j < n:
      j += 1
    i += 1
```

- ○ $O(1)$
- ● $O(n)$
- ○ $O(\log(n))$
- ○ $O(n\log(n))$
- ○ $O(n^2)$
- ○ $O(2^n)$

Solution: The outer loop runs `n` times. The inner loop runs `n` times, but it only runs once because `j` is *not* reset after each outer loop. Thus, the runtime is $O(n)$.

Q7.5 (1 point)

```
def f(n):
  if n == 0 or n == 1:
    return n
  return f(n // 2)
```

- ○ $O(1)$
- ○ $O(n)$
- ● $O(\log(n))$
- ○ $O(n\log(n))$
- ○ $O(n^2)$
- ○ $O(2^n)$

Solution: This is a recursive function that halves the input size `n` every time. This means it runs $O(\log_2(n))$ times, or simply $O(\log(n))$ times. For example, consider $n = 128$. Every time `f` is called, `n` becomes 64, 32, 16, 8, 4, 2, and finally 1 (the base case). $\log_2(128) = 7$, which is how many calls to `f` there were.

Q8 *Air88*

(13 points)

The Data C88C staff are going on summer break to touch some grass! Below are two tables representing connecting flights taken by various passengers.

For this problem, we define a **trip** to be made of two (connecting) **flights**.

- The `first_flight` table records the first connecting flight for each passenger.
- The `second_flight` table records their second connecting flight.
- The `pid` (passenger ID) column uniquely identifies each individual traveler.

pid	name	airline	start	end	international	duration
3791	Edwin	Alaska	San Diego	Tokyo	TRUE	11.5
3413	Dhruv	United	Chicago	San Francisco	FALSE	4
9221	Mike Baller	Southwest	Los Angeles	New York City	FALSE	5
9201	John Amongus	Spirit	Philadelphia	Pittsburgh	FALSE	1
0182	Mira	United	San Francisco	Tahiti	TRUE	9
2718	Alicia	American	Taipei	San Francisco	TRUE	11.5
0192	Grace B	American	Honolulu	Los Angeles	FALSE	5.5
6628	Reema	Delta	San Francisco	Los Angeles	FALSE	1.5
0987	Lia	United	San Francisco	Mexico City	TRUE	4.5
1029	Ramya	Southwest	San Francisco	Seattle	FALSE	2
6371	Isabelle	United	Los Angeles	London	TRUE	11
8888	Ethan	American	San Francisco	Los Angeles	FALSE	1.5
0123	Grace X	Southwest	San Francisco	New York City	FALSE	5.5
1231	Rebecca	United	Miami	San Francisco	FALSE	6

Table 1: `first_flight`

pid	start	end	international	duration
1231	San Francisco	Miami	FALSE	6
3791	Tokyo	Seoul	TRUE	2.5
9221	New York City	Los Angeles	FALSE	5
2718	San Francisco	Irvine	FALSE	1.5
3413	San Francisco	Honolulu	FALSE	5.5
0192	Los Angeles	Honolulu	FALSE	5.5
6628	Los Angeles	San Francisco	FALSE	1.5
0182	Tahiti	Honolulu	TRUE	6
0987	Mexico City	San Francisco	TRUE	4.5
1029	Seattle	Vancouver	TRUE	1
6371	London	San Francisco	TRUE	11
8888	Los Angeles	Bali	TRUE	17
0123	New York City	San Francisco	FALSE	5.5
9201	Pittsburgh	Phoenix	FALSE	4.5

Table 2: `second_flight`

Q8.1 (1 point) If we grouped `second_flight` flights by whether they were **international** or not, how many rows would our resulting table output contain?

2

Q8.2 - Q8.5 (4 points) Complete the SQL query to return passengers' full trips whose **initial starting point** does **NOT** match their **final destination** after the connecting flight. Your query should return the following columns: `pid`, `name`, `start` (the start of the **first** flight), and `end` (the end of the **second** flight).

You should expect the following output:

pid	name	start	end
0182	Mira	San Francisco	Honolulu
1029	Ramya	San Francisco	Vancouver
2718	Alicia	Taipei	Irvine
3413	Dhruv	Chicago	Honolulu
3791	Edwin	San Diego	Seoul
6371	Isabelle	Los Angeles	San Francisco
8888	Ethan	San Francisco	Bali
9201	John Amongus	Philadelphia	Phoenix

Note: There are multiple possible solutions, one of which does not require you to use all blanks. We will accept all of them.

```
1 SELECT first.pid, first.name, first.start, second.end
                                  Q8.2
2 FROM first_flight AS first

3 JOIN second_flight AS second
            Q8.3
4   ON first.pid = second_flight.pid
                     Q8.4

5 WHERE first.start != second.end;
                    Q8.5
```

Solution: An alternate solution is to omit the `WHERE` clause and include the `WHERE` predicate in the `ON` clause, e.g.:

```
SELECT first.pid, first.name, first.start, second.end
FROM first_flight AS first
JOIN second_flight AS second
  ON first.pid = second.pid AND first.start != second.end;
```

Q8.6 - Q8.9 (4 points) Complete the SQL query that, only considering trips initially flying out of San Francisco, outputs the names of passengers and their total flight time. Sort the result by total flight time in descending order. Your result should include: `name` and `trip_length` (total time spent flying for the entire trip).

You should expect the following output:

name	trip_length
Ethan	18.5
Mira	15
Grace X	11.0
Lia	9.0
Ramya	3
Reema	3.0

```
1 SELECT first.name, first.duration + second.duration AS trip_length
                                                    Q8.6
2 FROM first_flight AS first, second_flight AS second
                                          Q8.7
3 WHERE first.pid = second.pid AND first.start = "San Francisco"
                                          Q8.8
4 ORDER BY trip_length DESC;
                 Q8.9
```

Q8.10 (1 point) Which of the following `WHERE` clauses would correctly match values **that contain a space in between words** (e.g., `"San Francisco"` or `"John Amongus"`)?

- ● `WHERE column LIKE '% %'`
- ○ `WHERE column LIKE '%'`
- ○ `WHERE column LIKE '%%'`
- ○ `WHERE column LIKE ' % '`

Q8.11 - Q8.13 (3 points) Complete the SQL query to find, for each **airline**, the number of **first connecting flights** that flew internationally. The output should only include airlines with at least 2 international first connecting flights.

You should expect the following output:

airline	COUNT(*)
United	3

```
1 SELECT airline, COUNT(*)
2 FROM first_flight

3 WHERE international
            Q8.11

4 GROUP BY airline
            Q8.12

5 HAVING COUNT(*) >= 2;
            Q8.13
```

Q9 *Just for fun!* **(0 points)**

Q9.1 Draw something fun, or write a message for the staff!